

Lecture 4

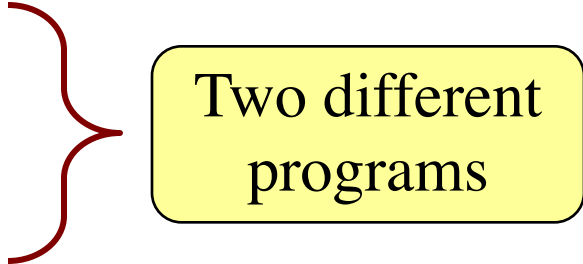
Defining Functions

Academic Integrity Quiz

- **Remember:** quiz about the course AI policy
 - Have posted grades for completed quizzes
 - Right now, missing ~130 enrolled students
 - If did not receive at least 9/10, take it again
- If you are not aware of the quiz
 - Go to <http://www.cs.cornell.edu/courses/cs11110/>
 - Click **Academic Integrity** in side bar
 - Read and take quiz in CMS

Recall: Modules

- Modules provide extra functions, variables
 - **Example:** math provides `math.cos()`, `math.pi`
 - Access them with the `import` command
- Python provides a lot of them for us
- **This Lecture:** How to make modules
 - Atom Editor to *make* a module
 - Python to *use* the module



Two different programs

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
>>> plus(23)
```

```
24
```

```
>>>
```

Function Definition

- Defines what function **does**

```
def plus(n):
```

```
    return n+1
```

- **Parameter:** variable that is listed within the parentheses of a method header.
- **Argument:** a value to assign to the method parameter when it is called

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
>>> plus(23)
```

```
24
```

```
>>>
```

Function
Header

Function Definition

- Defines what function **does**

```
def plus(n):
```

```
    return n+1
```

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
>>> plus(23)
```

```
24
```

```
>>>
```

Function
Header

```
def plus(n):
```

```
    return n+1
```

Function
Body
(indented)

- **Parameter:** variable that is listed within the parentheses of a method header.
- **Argument:** a value to assign to the method parameter when it is called

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
>>> plus(23)
```

```
24
```

argument to
assign to n

Function Definition

- Defines what function **does**

Function
Header

```
def plus(n):
```

```
    return n+1
```

declaration of
parameter n

Function
Body
(indented)

- **Parameter:** variable that is listed within the parentheses of a method header.
- **Argument:** a value to assign to the method parameter when it is called

Anatomy of a Function Definition

name

parameters

```
def plus(n):
```

Function Header

```
    """Returns the number n+1
```

Docstring
Specification

```
    Parameter n: number to add to  
    Precondition: n is a number"""
```

```
    x = n+1
```

Statements to
execute when called

```
    return x
```


Anatomy of a Function Definition

name

parameters

```
def plus(n):
```

Function Header

```
    """Returns the number n+1
```

Docstring
Specification

```
    Parameter n: number to add to  
    Precondition: n is a number"""
```

```
    x = n+1
```

Statements to
execute when called

```
    return x
```

The vertical line
indicates indentation

Use vertical lines when you write Python
on **exams** so we can see indentation

The **return** Statement

- **Format:** `return <expression>`
 - Used to evaluate *function call* (as an expression)
 - Also stops executing the function!
 - Any statements after a **return** are ignored
- **Example:** temperature converter function

```
def to_centiGrade(x):
```

```
    """Returns: x converted to centigrade"""
```

```
    return 5*(x-32)/9.0
```

A More Complex Example


Function Definition

```
def foo(a,b):  
    """Return something  
    Param a: number  
    Param b: number"""  
  
    x = a  
    y = b  
    return x*y+y
```

Function Call

```
>>> x = 2
```

```
>>> foo(3,4)
```

x 

What is in the box?

A More Complex Example

Function Definition

```
def foo(a,b):  
    """Return something  
    Param a: number  
    Param b: number"""  
  
    x = a  
    y = b  
    return x*y+y
```

Function Call

```
>>> x = 2
```

```
>>> foo(3,4)
```

x ?

What is in the box?

A: 2

B: 3

C: 16

D: Nothing!

E: I do not know

A More Complex Example

Function Definition

```
def foo(a,b):  
    """Return something  
    Param a: number  
    Param b: number"""  
  
    x = a  
    y = b  
    return x*y+y
```

Function Call

```
>>> x = 2
```

```
>>> foo(3,4)
```

x ?

What is in the box?

A: 2 **CORRECT**

B: 3

C: 16

D: Nothing!

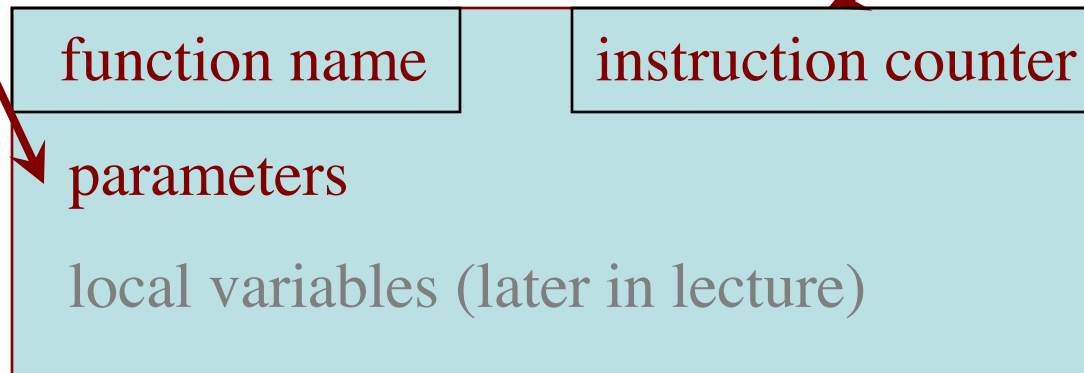
E: I do not know

Understanding How Functions Work

- **Function Frame:** Representation of function call
- A **conceptual model** of Python

Draw parameters
as variables
(named boxes)

- Number of statement in the
function body to execute next
- **Starts with 1**



Text (Section 3.10) vs. Class

Textbook

to_centrigrade

$x \rightarrow 50.0$

This Class

to_centrigrade

1

x 50.0

Definition:

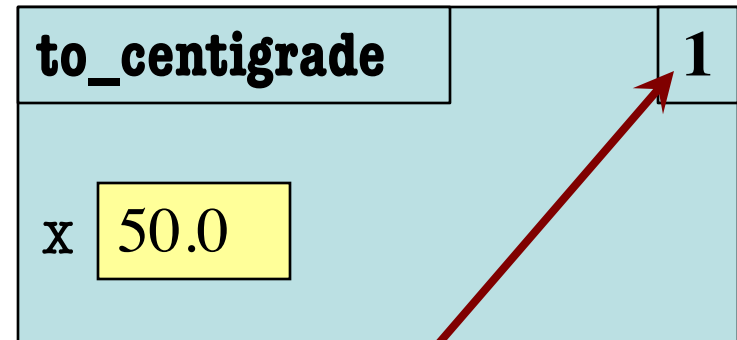
```
def to_centrigrade(x):  
    | return 5*(x-32)/9.0
```

Call: to_centrigrade(50.0)

Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
 - Look for variables in the frame
 - If not there, look for global variables with that name
4. Erase the frame for the call

Initial call frame
(before exec body)



next line to execute

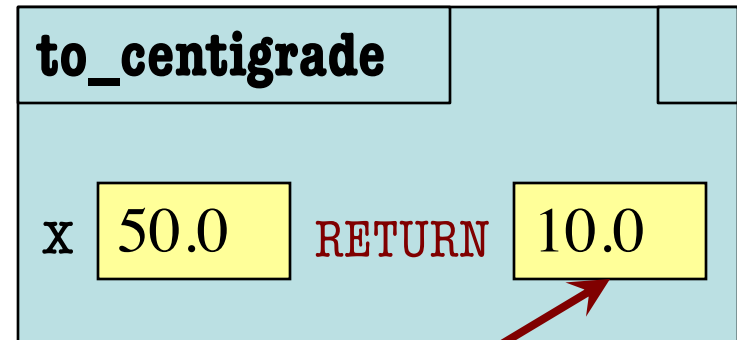
```
def to_centigrade(x):  
1 | return 5*(x-32)/9.0
```


Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
 - Look for variables in the frame
 - If not there, look for global variables with that name
4. Erase the frame for the call

```
1 def to_centigrade(x):  
  | return 5*(x-32)/9.0
```

Executing the
return statement



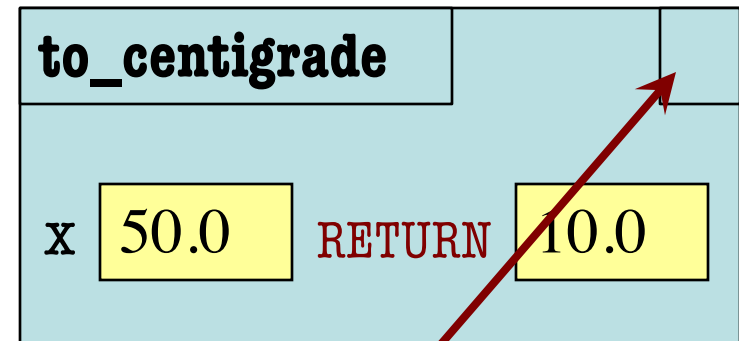
Return statement creates a
special variable for result

Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
 - Look for variables in the frame
 - If not there, look for global variables with that name
4. Erase the frame for the call

```
def to_centigrade(x):  
1 | return 5*(x-32)/9.0
```

Executing the
return statement



The return terminates;
no next line to execute

Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
 - Look for variables in the frame
 - If not there, look for global variables with that name
4. Erase the frame for the call

ERASE WHOLE FRAME

```
1 def to_centigrade(x):  
  | return 5*(x-32)/9.0
```

But don't actually
erase on an exam

Call Frames vs. Global Variables

The specification is a **lie**:

```
def swap(a,b):  
    """Swap global a & b"""  
    1   tmp = a  
    2   a = b  
    3   b = tmp
```

```
>>> a = 1
```

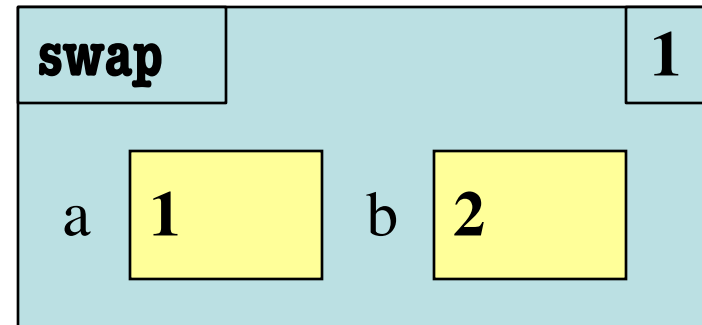
```
>>> b = 2
```

```
>>> swap(a,b)
```

Global Variables

a **1** b **2**

Call Frame



Call Frames vs. Global Variables

The specification is a **lie**:

```
def swap(a,b):  
    """Swap global a & b"""  
    1   tmp = a  
    2   a = b  
    3   b = tmp
```

```
>>> a = 1
```

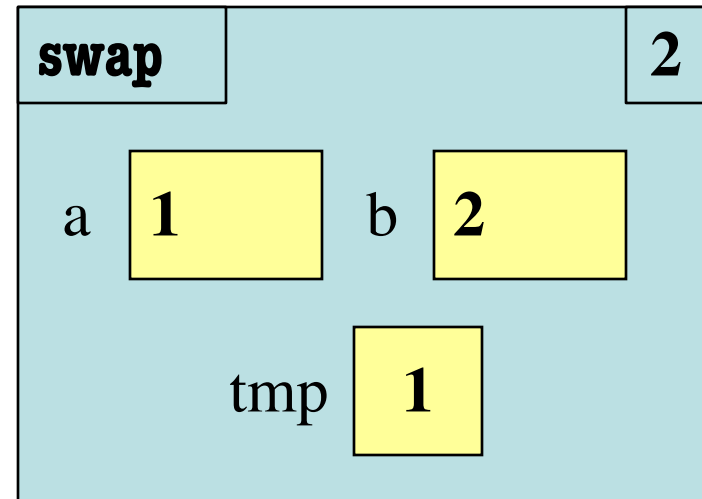
```
>>> b = 2
```

```
>>> swap(a,b)
```

Global Variables

a **1** b **2**

Call Frame



Call Frames vs. Global Variables

The specification is a **lie**:

```
def swap(a,b):  
    """Swap global a & b"""  
    1 tmp = a  
    2 a = b  
    3 b = tmp
```

```
>>> a = 1
```

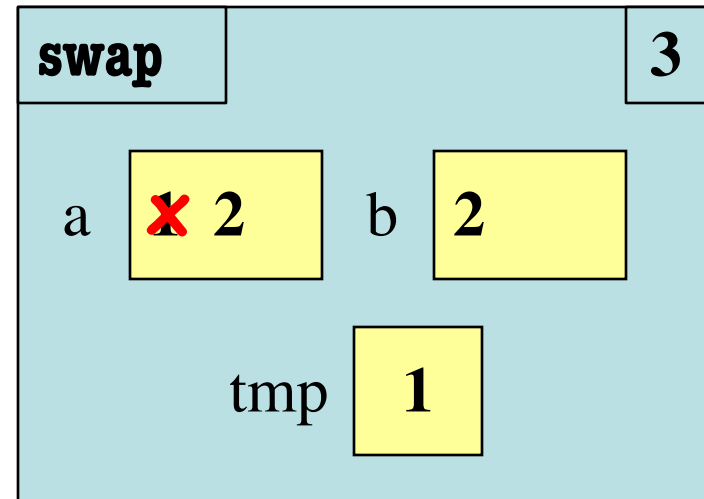
```
>>> b = 2
```

```
>>> swap(a,b)
```

Global Variables

a **1** b **2**

Call Frame



Call Frames vs. Global Variables

The specification is a **lie**:

```
def swap(a,b):  
    """Swap global a & b"""  
    1 tmp = a  
    2 a = b  
    3 b = tmp
```

```
>>> a = 1
```

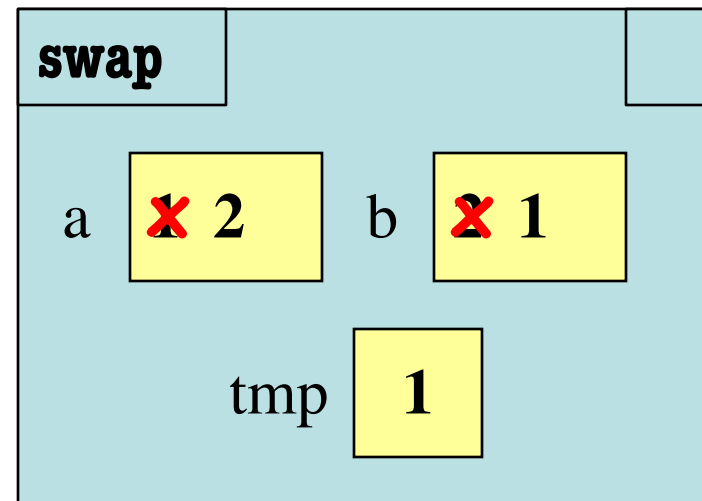
```
>>> b = 2
```

```
>>> swap(a,b)
```

Global Variables

a **1** b **2**

Call Frame



Call Frames vs. Global Variables

The specification is a **lie**:

```
def swap(a,b):  
    """Swap global a & b"""  
1   tmp = a  
2   a = b  
3   b = tmp
```

```
>>> a = 1
```

```
>>> b = 2
```

```
>>> swap(a,b)
```

Global Variables

a 1 b 2

Call Frame

ERASE THE FRAME

Exercise Time

Function Definition

```
def foo(a,b):
```

```
    """Return something  
    Param x: a number  
    Param y: a number"""
```

```
1 x = a
```

```
2 y = b
```

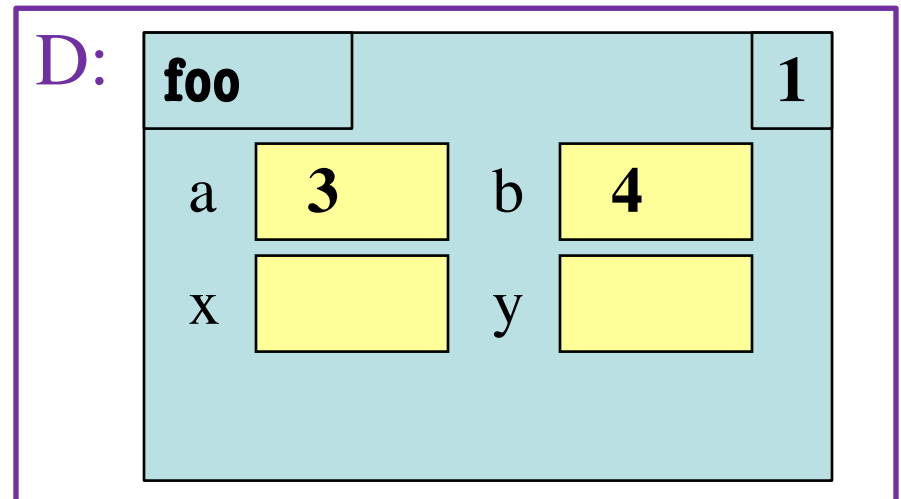
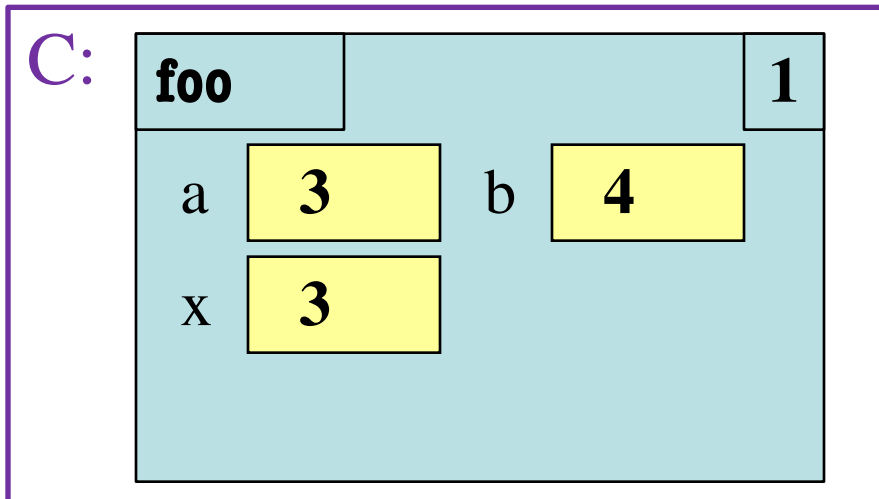
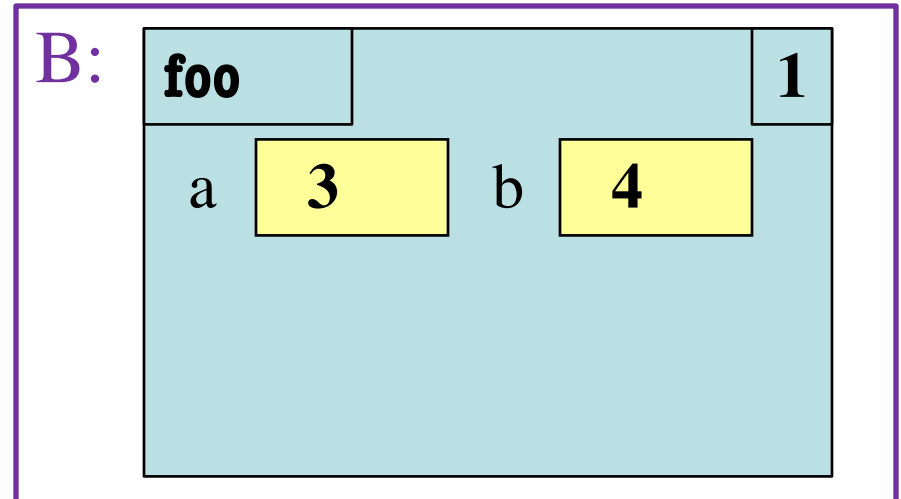
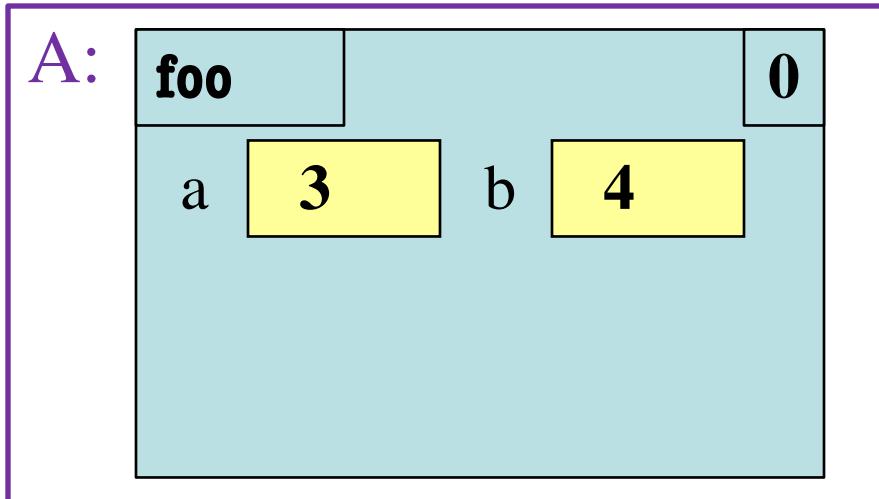
```
3 return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

What does the
frame look like
at the **start**?

Which One is Closest to Your Answer?



Which One is Closest to Your Answer?

A:

foo		0	
a	3	b	4

B:

foo		1	
a	3	b	4

E: $_ _ (_) _ / _$

C:

foo		
a	3	
x	3	

D:

		1	
b	4		
x		y	

Exercise Time

Function Definition

```
def foo(a,b):
```

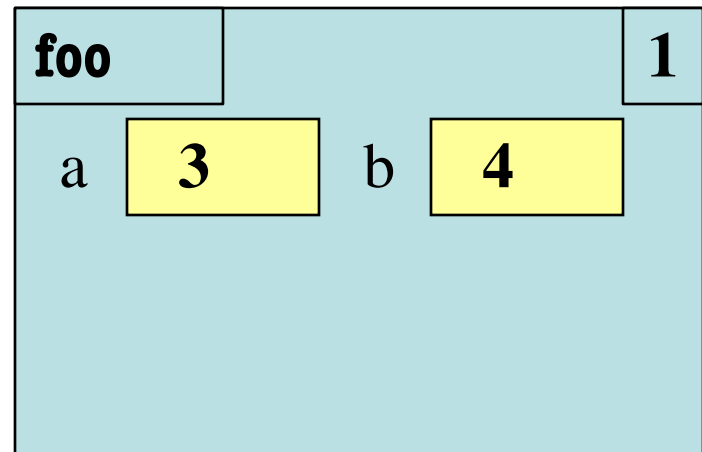
```
    """Return something  
    Param x: a number  
    Param y: a number"""
```

```
1  x = a  
2  y = b  
3  return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

B:



Exercise Time

Function Definition

```
def foo(a,b):
```

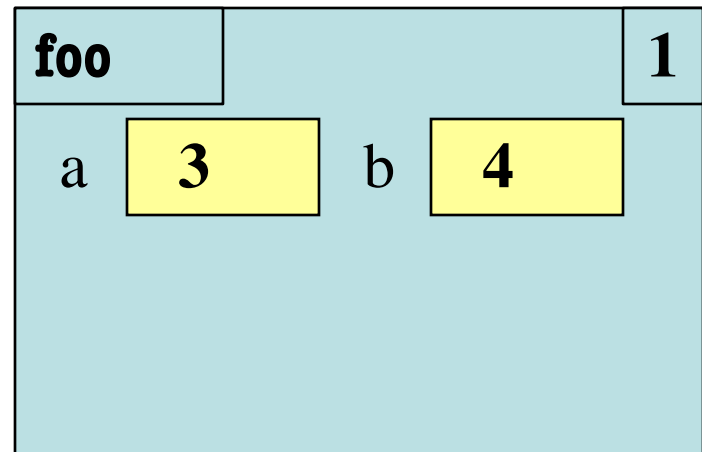
```
    """Return something  
    Param x: a number  
    Param y: a number"""
```

```
1   x = a  
2   y = b  
3   return x*y+y
```

Function Call

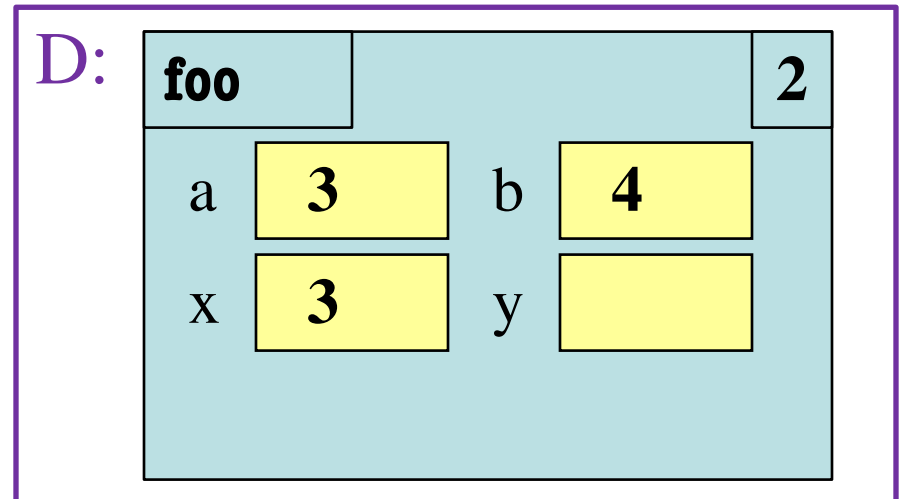
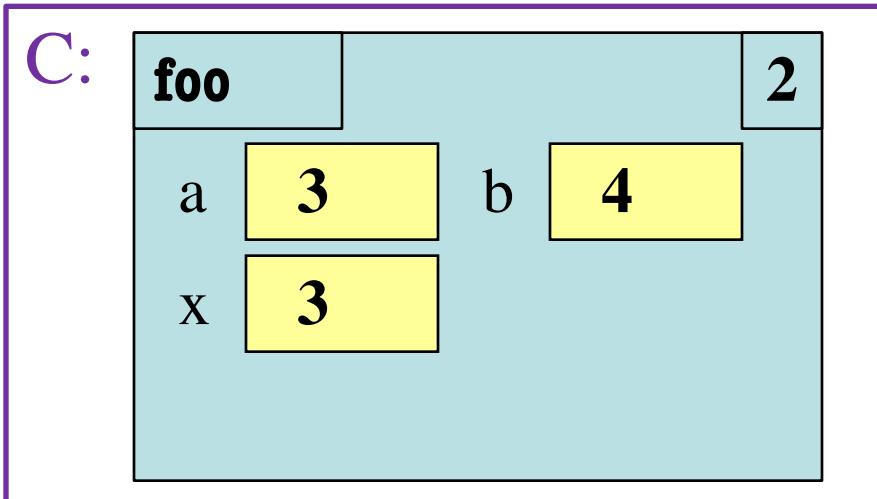
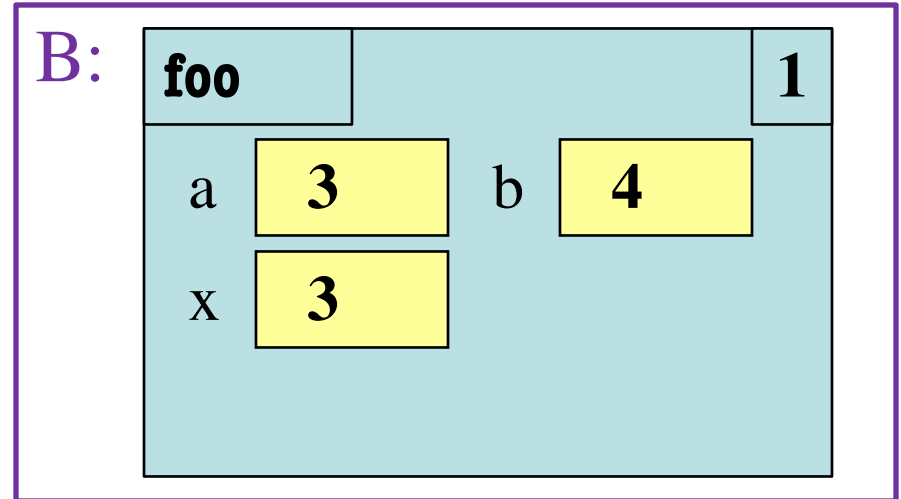
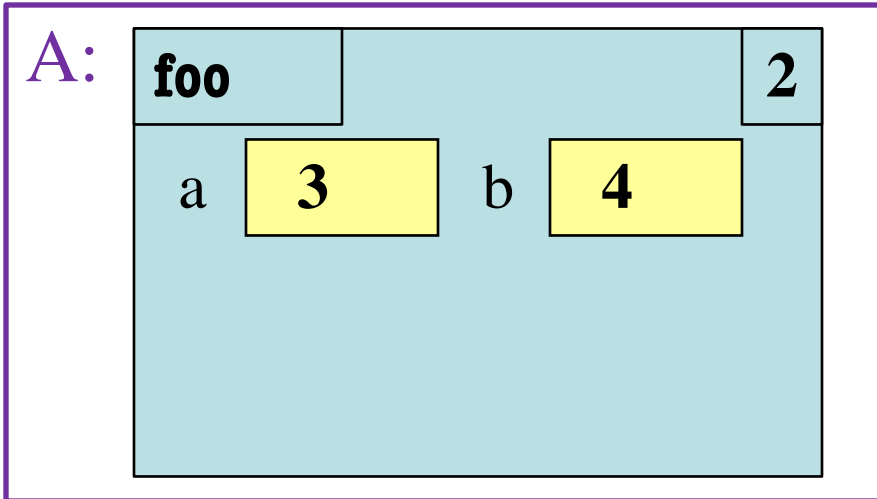
```
>>> x = foo(3,4)
```

B:



What is the **next step**?

Which One is Closest to Your Answer?



Exercise Time

Function Definition

```
def foo(a,b):
```

```
    """Return something
```

```
       Param x: a number
```

```
       Param y: a number"""
```

```
1   x = a
```

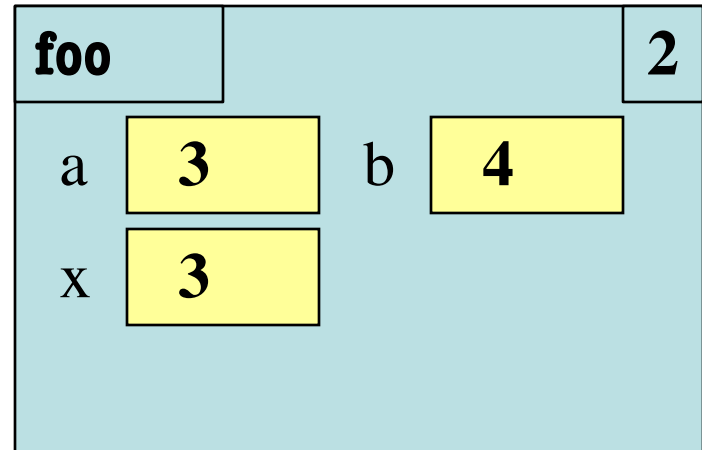
```
2   y = b
```

```
3   return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

C:



Exercise Time

Function Definition

```
def foo(a,b):
```

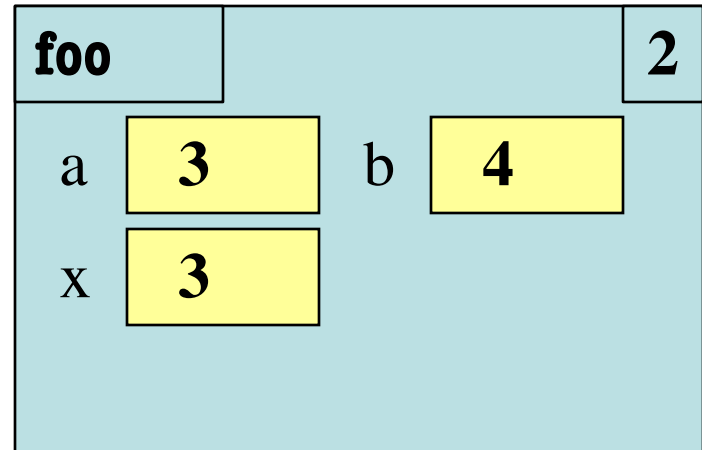
```
    """Return something  
    Param x: a number  
    Param y: a number"""
```

```
1 x = a  
2 y = b  
3 return x*y+y
```

Function Call

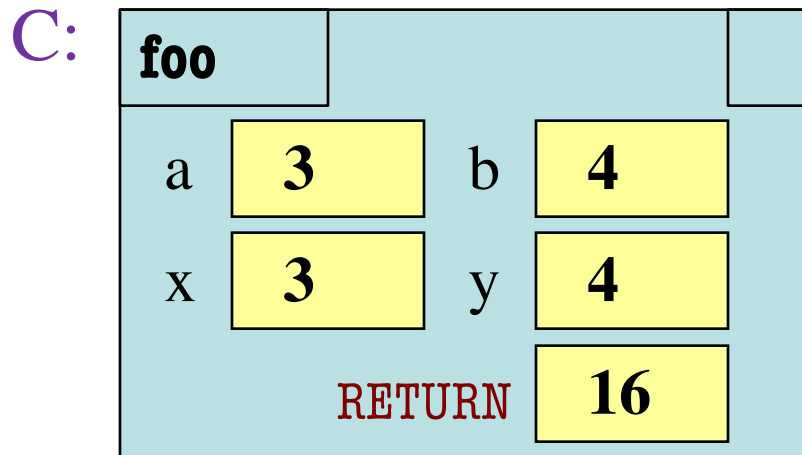
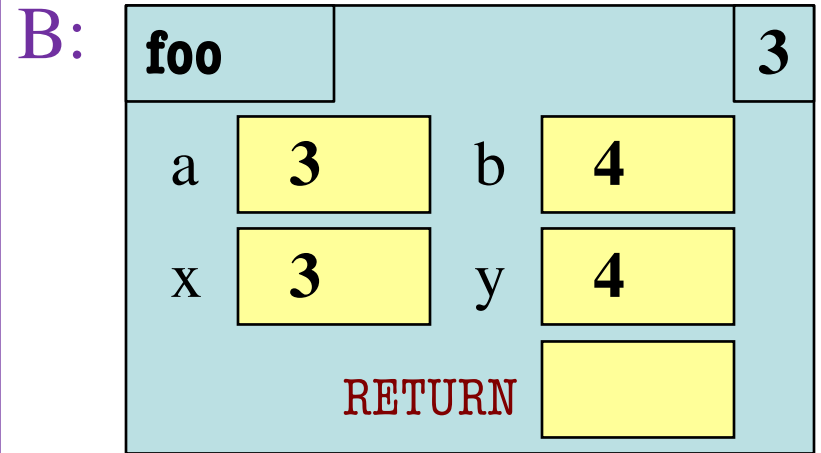
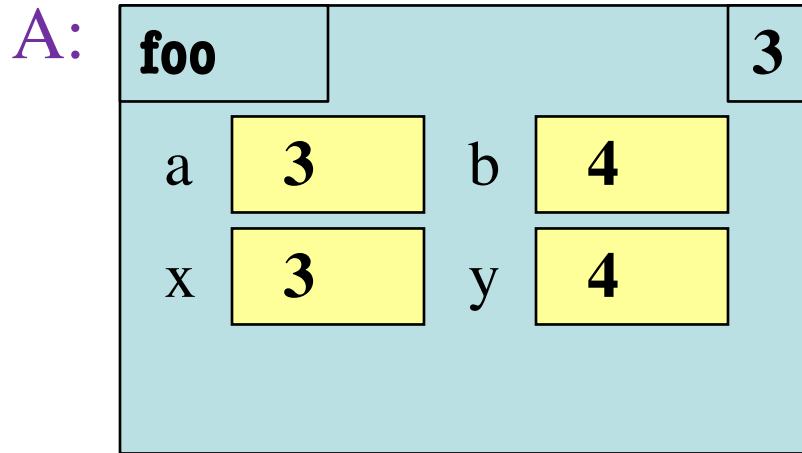
```
>>> x = foo(3,4)
```

C:



What is the **next step**?

Which One is Closest to Your Answer?



Exercise Time

Function Definition

```
def foo(a,b):
```

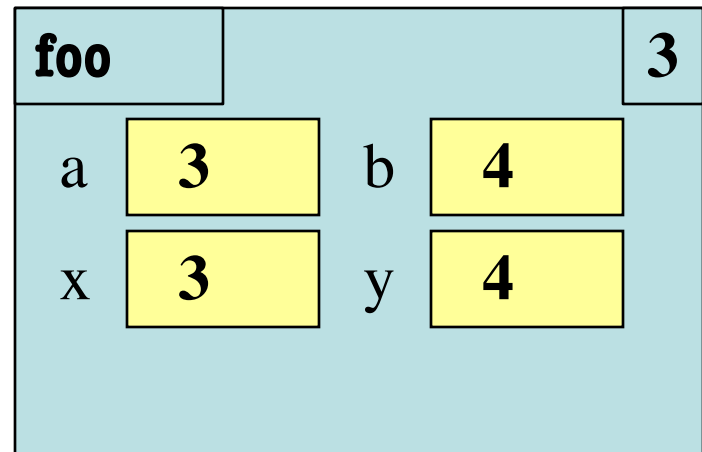
```
    """Return something  
    Param x: a number  
    Param y: a number"""
```

```
1  x = a  
2  y = b  
3  return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

A:



Exercise Time

Function Definition

```
def foo(a,b):
```

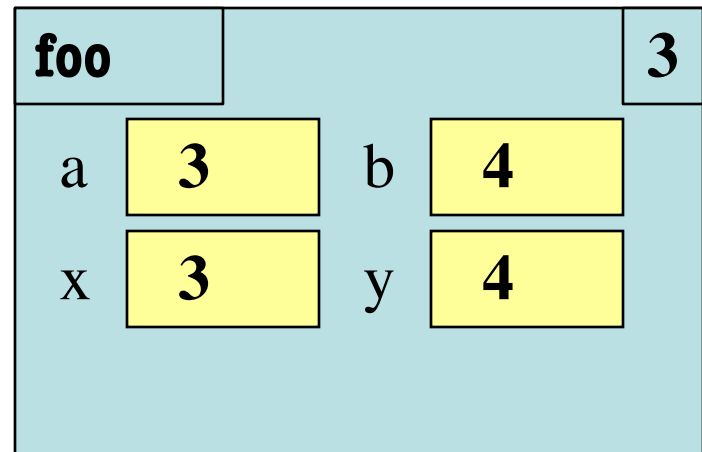
```
    """Return something  
    Param x: a number  
    Param y: a number"""
```

```
1  x = a  
2  y = b  
3  return x*y+y
```

Function Call

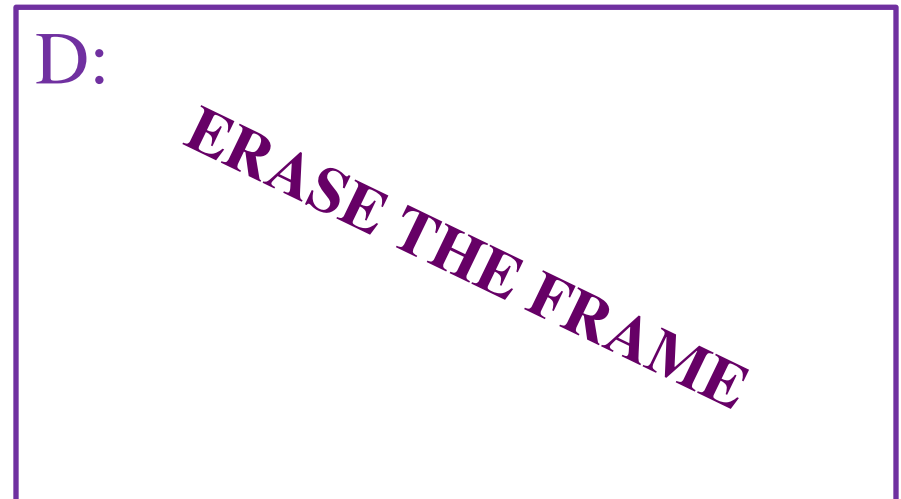
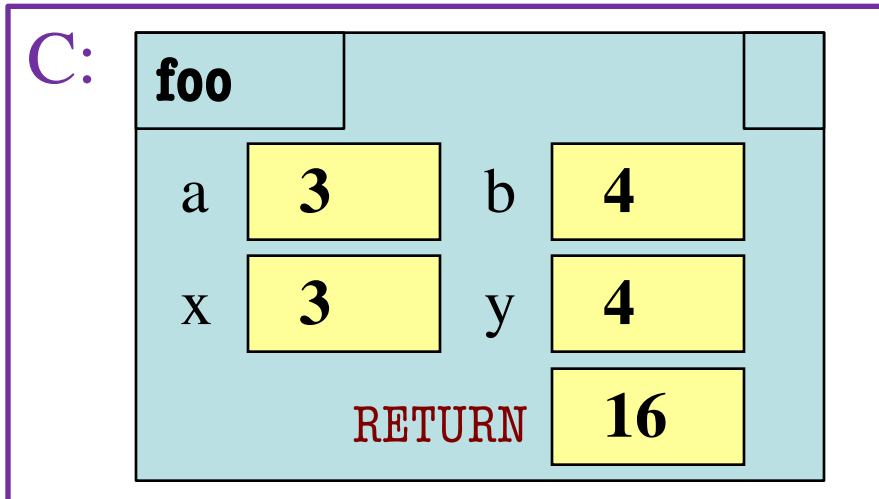
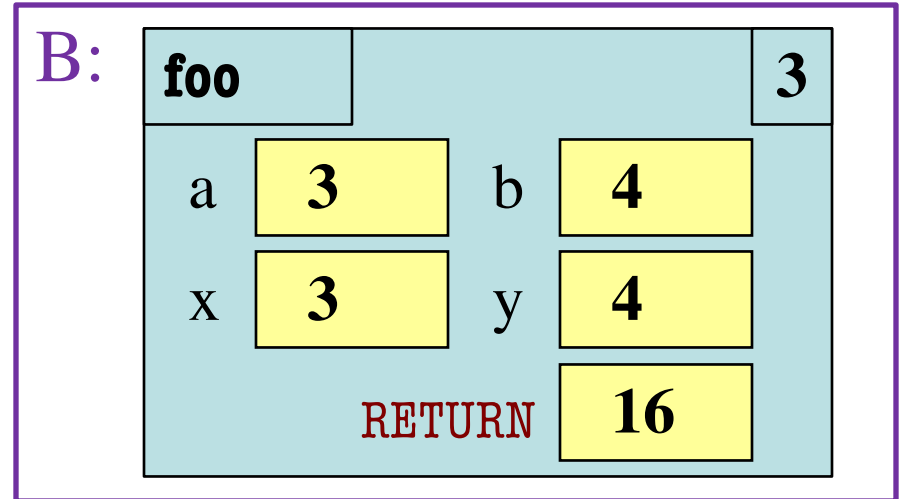
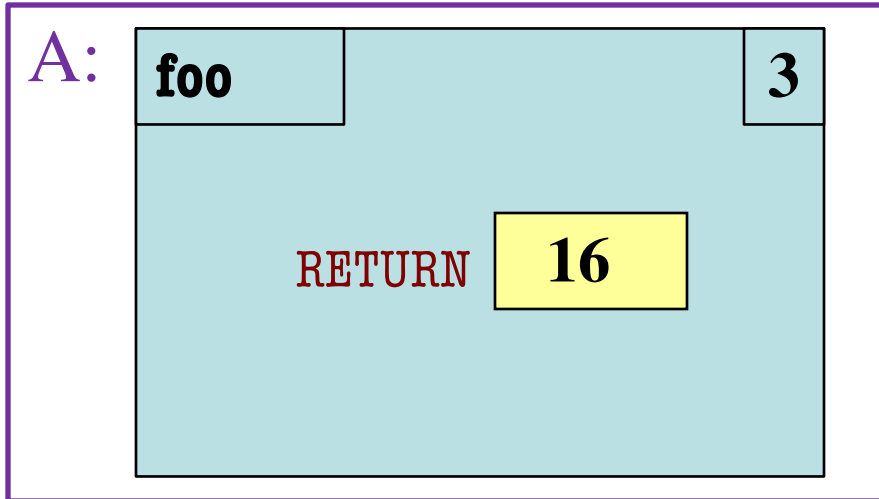
```
>>> x = foo(3,4)
```

A:



What is the **next step**?

Which One is Closest to Your Answer?



Exercise Time

Function Definition

```
def foo(a,b):
```

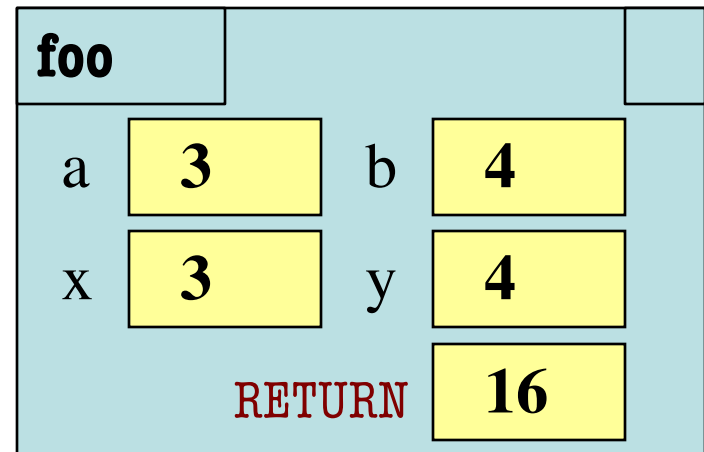
```
    """Return something  
    Param x: a number  
    Param y: a number"""
```

```
1 x = a  
2 y = b  
3 return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

C:



Exercise Time

Function Definition

```
def foo(a,b):
```

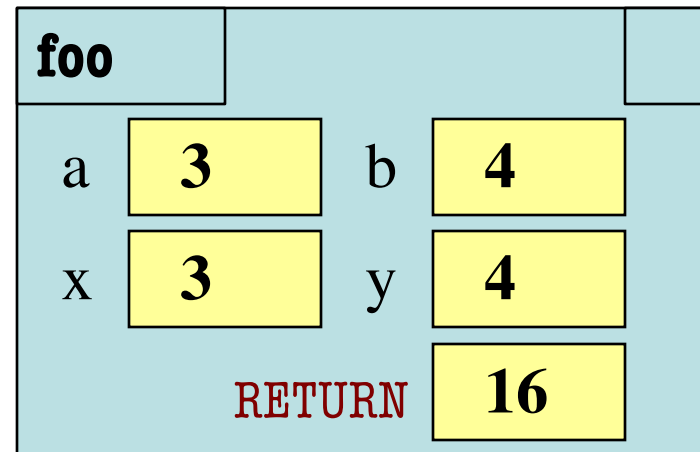
```
    """Return something  
    Param x: a number  
    Param y: a number"""
```

```
1 x = a  
2 y = b  
3 return x*y+y
```

Function Call

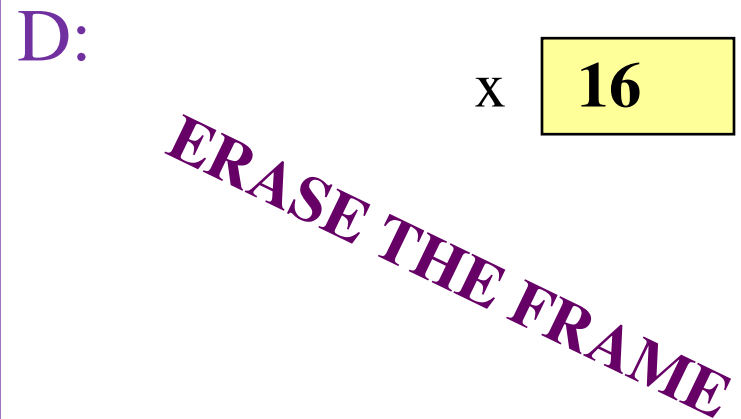
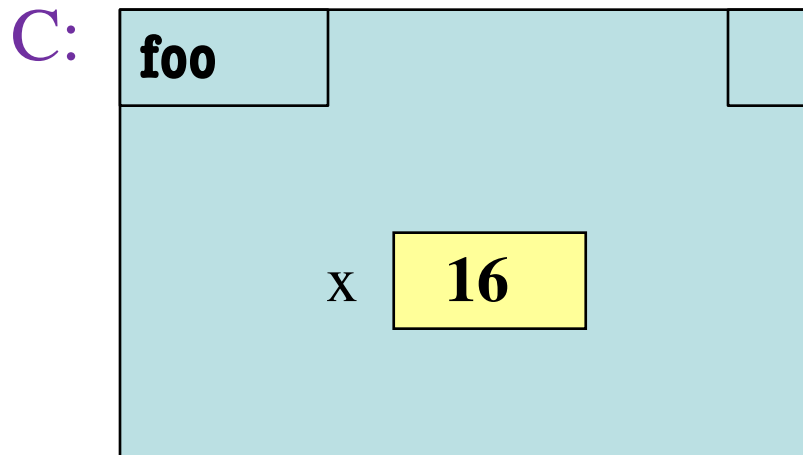
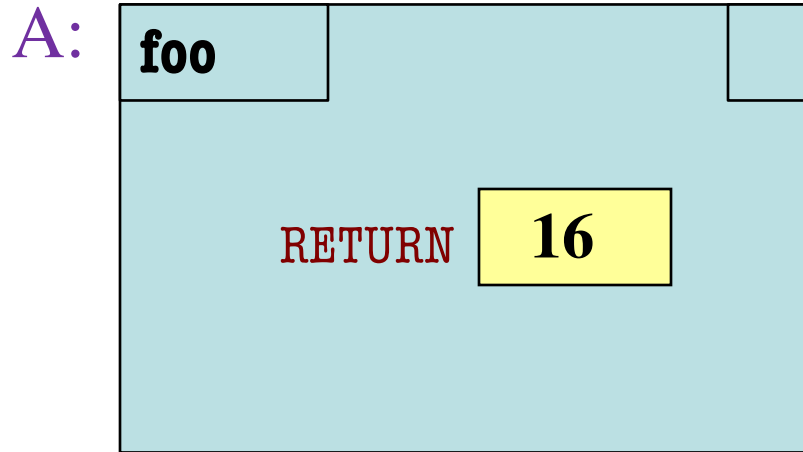
```
>>> x = foo(3,4)
```

C:



What is the **next step**?

Which One is Closest to Your Answer?



Exercise Time

Function Definition

```
def foo(a,b):  
    """Return something  
    Param x: a number  
    Param y: a number"""  
1   x = a  
2   y = b  
3   return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

D:

x

16

ERASE THE FRAME

Exercise Time

Function Definition

```
def foo(a,b):  
    """Return something  
    Param x: a number  
    Param y: a number"""  
1   x = a  
2   y = b  
3   return x*y+y
```

Function Call

```
>>> x = foo(3,4)
```

D:

Variable in
global space

x

16

ERASE THE FRAME

Visualizing Frames: The Python Tutor

```
→ 1 def max(x,y):  
  2     if x > y:  
  3         return x  
  4     return y  
  5  
  6 a = 1  
  7 b = 2  
→ 8 max(a,b)
```

[Edit code](#)

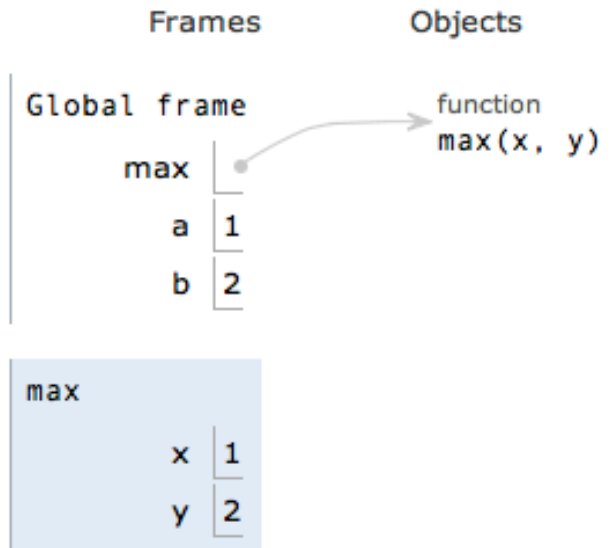
<< First

< Back

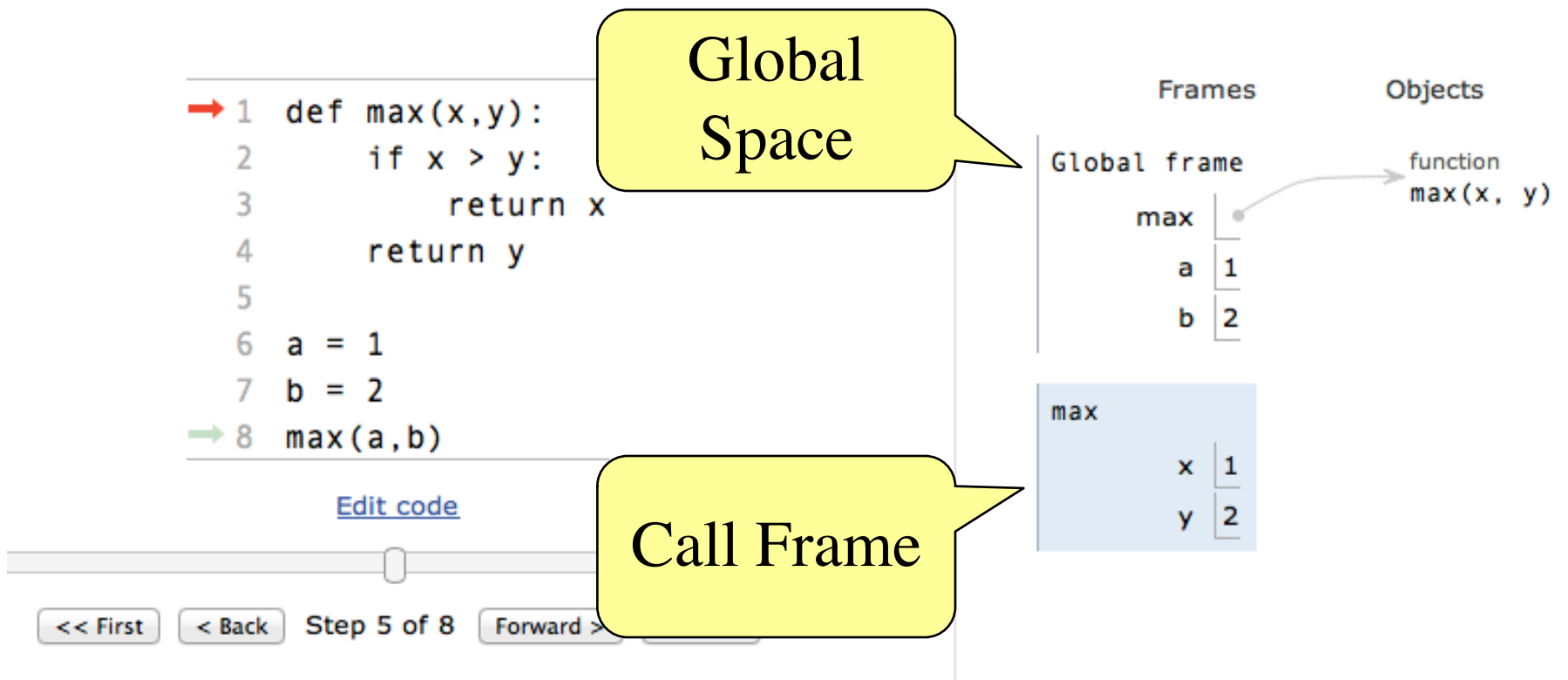
Step 5 of 8

Forward >

Last >>



Visualizing Frames: The Python Tutor



Visualizing Frames: The Python Tutor

The image shows a Python Tutor interface. On the left, a code editor displays the following code:

```
1 def max(x,y):  
2     if x > y:  
3         return x  
4     return y  
5  
6 a = 1  
7 b = 2  
8 max(a,b)
```

Below the code is an "Edit code" link and a progress bar showing "Step 5 of 8". Navigation buttons include "<< First", "< Back", "Forward >", and ">> Last".

On the right, the "Global Frame" is visualized as a table:

max	function max(x, y)
a	1
b	2

An arrow points from the "max" entry in the Global Frame to the function definition in the code editor.

Below the Global Frame, the "Call Frame" for the function "max" is visualized as a table:

x	1
y	2

Two yellow callout boxes are present: one labeled "Global Space" pointing to the Global Frame, and another labeled "Call Frame" pointing to the Call Frame. A green callout box labeled "Variables from second lecture go in here" points to the Global Frame.

Visualizing Frames: The Python Tutor

```
→ 1 def max(x,y):  
  2     if x > y:  
  3         return x  
  4     return y  
  5  
  6 a = 1  
  7 b = 2  
→ 8 max(a,b)
```

[Edit code](#)

<< First

< Back

Step 5 of 8

Forward >

Last >>

Frames

Objects

Global fr

max

Missing line numbers!

a

b

max

x

1

y

2

Visualizing Frames: The Python Tutor

Line number
marked here
(sort-of)

```
→ 1 def max(x,y):  
  2     if x > y:  
  3         return x  
  4     return y  
  5  
  6 a = 1  
  7 b = 2  
→ 8 max(a,b)
```

[Edit code](#)

<< First

< Back

Step 5 of 8

Forward >

Last >>

Frames

Objects

Global fr

max

Missing line
numbers!

a

b

max

x | 1

y | 2

Next Time: Text Processing